```
function [traj, infStates] = tapas_hgf_binary(r, p, varargin)
% Calculates the trajectories of the agent's representations under the
 HGF
%
% This function can be called in two ways:
%
% (1) tapas_hgf_binary(r, p)
%
%     where r is the structure generated by tapas_fitModel and p is
 the parameter vector in native space;
%
% (2) tapas_hgf_binary(r, ptrans, 'trans')
%
%     where r is the structure generated by tapas_fitModel, ptrans is
 the parameter vector in
%     transformed space, and 'trans' is a flag indicating this.
%
%
 ---------------------------------------------------------------------------------
% Copyright (C) 2012-2017 Christoph Mathys, TNU, UZH & ETHZ
%
% This file is part of the HGF toolbox, which is released under the
 terms of the GNU General Public
% Licence (GPL), version 3. You can redistribute it and/or modify it
 under the terms of the GPL
% (either version 3 or, at your option, any later version). For
 further details, see the file
% COPYING or <http://www.gnu.org/licenses/>.


% Transform paramaters back to their native space if needed
if ~isempty(varargin) && strcmp(varargin{1},'trans');
    p = tapas_hgf_binary_transp(r, p);
end

% Number of levels
try
    l = r.c_prc.n_levels;
catch
    l = (length(p)+1)/5;

    if l ~= floor(l)
        error('tapas:hgf:UndetNumLevels', 'Cannot determine number of
 levels');
    end
end

% Unpack parameters
mu_0 = p(1:l);
sa_0 = p(l+1:2*l);
rho  = p(2*l+1:3*l);
ka   = p(3*l+1:4*l-1);
```

```
om    = p(4*l:5*l-2);
th    = exp(p(5*l-1));

% Add dummy "zeroth" trial
u = [0; r.u(:,1)];

% Number of trials (including prior)
n = length(u);

% Assume that if u has more than one column, the last contains t
try
    if r.c_prc.irregular_intervals
        if size(u,2) > 1
            t = [0; r.u(:,end)];
        else
            error('tapas:hgf:InputSingleColumn', 'Input matrix
 must contain more than one column if irregular_intervals is set to
 true.');
        end
    else
        t = ones(n,1);
    end
catch
    if size(u,2) > 1
        t = [0; r.u(:,end)];
    else
        t = ones(n,1);
    end
end

% Initialize updated quantities

% Representations
mu = NaN(n,l);
pi = NaN(n,l);

% Other quantities
muhat = NaN(n,l);
pihat = NaN(n,l);
v     = NaN(n,l);
w     = NaN(n,l-1);
da    = NaN(n,l);

% Representation priors
% Note: first entries of the other quantities remain
% NaN because they are undefined and are thrown away
% at the end; their presence simply leads to consistent
% trial indices.
mu(1,1) = tapas_sgm(mu_0(1), 1);
pi(1,1) = Inf;
mu(1,2:end) = mu_0(2:end);
pi(1,2:end) = 1./sa_0(2:end);

% Pass through representation update loop
```

```
for k = 2:1:n
    if not(ismember(k-1, r.ign))

        %%%%%%%%%%%%%%%%%%%%%%
        % Effect of input u(k)
        %%%%%%%%%%%%%%%%%%%%%%

        % 2nd level prediction
        muhat(k,2) = mu(k-1,2) +t(k) *rho(2);

        % 1st level
        % ~~~~~~~~~
        % Prediction
        muhat(k,1) = tapas_sgm(ka(1) *muhat(k,2), 1);

        % Precision of prediction
        pihat(k,1) = 1/(muhat(k,1)*(1 -muhat(k,1)));

        % Updates
        pi(k,1) = Inf;
        mu(k,1) = u(k);

        % Prediction error
        da(k,1) = mu(k,1) -muhat(k,1);

        % 2nd level
        % ~~~~~~~~~
        % Prediction: see above

        % Precision of prediction
        pihat(k,2) = 1/(1/pi(k-1,2) +exp(ka(2) *mu(k-1,3) +om(2)));

        % Updates
        pi(k,2) = pihat(k,2) +ka(1)^2/pihat(k,1);
        mu(k,2) = muhat(k,2) +ka(1)/pi(k,2) *da(k,1);

        % Volatility prediction error
        da(k,2) = (1/pi(k,2) +(mu(k,2) -muhat(k,2))^2) *pihat(k,2) -1;

        if l > 3
            % Pass through higher levels
            % ~~~~~~~~~~~~~~~~~~~~~~~~~~
            for j = 3:l-1
                % Prediction
                muhat(k,j) = mu(k-1,j) +t(k) *rho(j);

                % Precision of prediction
                pihat(k,j) = 1/(1/pi(k-1,j) +t(k) *exp(ka(j) *mu(k-1,j
+1) +om(j)));

                % Weighting factor
                v(k,j-1) = t(k) *exp(ka(j-1) *mu(k-1,j) +om(j-1));
                w(k,j-1) = v(k,j-1) *pihat(k,j-1);
```

```
               % Updates
               pi(k,j) = pihat(k,j) +1/2 *ka(j-1)^2 *w(k,j-1)
*(w(k,j-1) +(2 *w(k,j-1) -1) *da(k,j-1));

               if pi(k,j) <= 0
                   error('tapas:hgf:NegPostPrec', 'Negative posterior
precision. Parameters are in a region where model assumptions are
violated.');
               end

               mu(k,j) = muhat(k,j) +1/2 *1/pi(k,j) *ka(j-1)
*w(k,j-1) *da(k,j-1);

               % Volatility prediction error
               da(k,j) = (1/pi(k,j) +(mu(k,j) -muhat(k,j))^2)
*pihat(k,j) -1;
           end
       end

       % Last level
       % ~~~~~~~~~~
       % Prediction
       muhat(k,l) = mu(k-1,l) +t(k) *rho(l);

       % Precision of prediction
       pihat(k,l) = 1/(1/pi(k-1,l) +t(k) *th);

       % Weighting factor
       v(k,l)   = t(k) *th;
       v(k,l-1) = t(k) *exp(ka(l-1) *mu(k-1,l) +om(l-1));
       w(k,l-1) = v(k,l-1) *pihat(k,l-1);

       % Updates
       pi(k,l) = pihat(k,l) +1/2 *ka(l-1)^2 *w(k,l-1) *(w(k,l-1) +(2
*w(k,l-1) -1) *da(k,l-1));

       if pi(k,l) <= 0
           error('tapas:hgf:NegPostPrec', 'Negative posterior
precision. Parameters are in a region where model assumptions are
violated.');
       end

       mu(k,l) = muhat(k,l) +1/2 *1/pi(k,l) *ka(l-1) *w(k,l-1)
*da(k,l-1);

       % Volatility prediction error
       da(k,l) = (1/pi(k,l) +(mu(k,l) -muhat(k,l))^2) *pihat(k,l) -1;
   else

       mu(k,:) = mu(k-1,:);
       pi(k,:) = pi(k-1,:);

       muhat(k,:) = muhat(k-1,:);
       pihat(k,:) = pihat(k-1,:);
```

```
        v(k,:)  = v(k-1,:);
        w(k,:)  = w(k-1,:);
        da(k,:) = da(k-1,:);

    end
end

% Implied learning rate at the first level
sgmmu2 = tapas_sgm(ka(1) *mu(:,2), 1);
dasgmmu2 = u -sgmmu2;
lr1    = diff(sgmmu2)./dasgmmu2(2:n,1);
lr1(da(2:n,1)==0) = 0;

% Remove representation priors
mu(1,:)  = [];
pi(1,:)  = [];

% Check validity of trajectories
if any(isnan(mu(:))) || any(isnan(pi(:)))
    error('tapas:hgf:VarApproxInvalid', 'Variational approximation
 invalid. Parameters are in a region where model assumptions are
 violated.');
else
    % Check for implausible jumps in trajectories
    dmu = diff(mu(:,2:end));
    dpi = diff(pi(:,2:end));
    rmdmu = repmat(sqrt(mean(dmu.^2)),length(dmu),1);
    rmdpi = repmat(sqrt(mean(dpi.^2)),length(dpi),1);

    jumpTol = 16;
    if any(abs(dmu(:)) > jumpTol*rmdmu(:)) || any(abs(dpi(:)) >
 jumpTol*rmdpi(:))
        error('tapas:hgf:VarApproxInvalid', 'Variational approximation
 invalid. Parameters are in a region where model assumptions are
 violated.');
    end
end

% Remove other dummy initial values
muhat(1,:) = [];
pihat(1,:) = [];
v(1,:)     = [];
w(1,:)     = [];
da(1,:)    = [];

% Create result data structure
traj = struct;

traj.mu     = mu;
traj.sa     = 1./pi;

traj.muhat  = muhat;
traj.sahat  = 1./pihat;
traj.v      = v;
traj.w      = w;
traj.da     = da;

% Updates with respect to prediction
traj.ud = mu -muhat;

% Psi (precision weights on prediction errors)
psi        = NaN(n-1,l);
psi(:,2)   = 1./pi(:,2);
psi(:,3:l) = pihat(:,2:l-1)./pi(:,3:l);
traj.psi   = psi;

% Epsilons (precision-weighted prediction errors)
epsi        = NaN(n-1,l);
epsi(:,2:l) = psi(:,2:l) .*da(:,1:l-1);
traj.epsi   = epsi;

% Full learning rate (full weights on prediction errors)
wt        = NaN(n-1,l);
wt(:,1)   = lr1;
wt(:,2)   = psi(:,2);
wt(:,3:l) = 1/2 *(v(:,2:l-1) *diag(ka(2:l-1))) .*psi(:,3:l);
traj.wt   = wt;

% Create matrices for use by the observation model
infStates = NaN(n-1,l,4);
infStates(:,:,1) = traj.muhat;
infStates(:,:,2) = traj.sahat;
infStates(:,:,3) = traj.mu;
infStates(:,:,4) = traj.sa;

return;
```

```
Not enough input arguments.

Error in tapas_hgf_binary (line 33)
    l = (length(p)+1)/5;
```

Published with MATLAB® R2018b